Letter Classification

Hannah Le

UCLA Department of Atmospheric and Oceanic Sciences

December 2024

Abstract

For this project, I've applied different types of supervised machine learning techniques including Logistic Regression, Random Forests, and Neural Networks (MLP) to classify different types of letters in the English alphabet. The results indicate that our Logistic Regression model struggled with correctly classifying each sample in our testing data, while our RF and MLP models performed well. We experiment with different model parameters in order to find the optimal amount of estimators and hidden layers in each of our respective models. Finally, we address over-fitting in our more complicated models by applying k-fold cross validation. In the end, we found that the Random Forest model performed the best overall due to its accuracy and convenience compared to the other models we tested.

1 Introduction

The classification and deciphering of handwritten letters is a fundamental and important task in optical character recognition (OCR). In recent years, this field has grown incredibly rapidly with applications in a variety a fields such as medicine[1] and teaching[2].
The intention of this project was to test and understand the effectiveness of different machine learning models applied to this letter recognition task. By analyzing our outcomes, we may be able to further improve this type of technology in these fields.

2 Data

This paper utilizes a dataset provided by the UCI Machine Learning Repository titled "Letter Recognition"[3]. This dataset contains 20,000 instances of generated capital letters based from 20 different fonts that were then randomly distorted to produce unique representations of each capital letter. Instead of using a classical approach utilizing computer vision, this dataset uses a unique approach that transforms each image into 16 numerical features including:

1. x-box: horizontal position of box
2. y-box: vertical position of box
3. width: width of box
4. high: height of box
5. onpix: total number of on pixels
6. x-bar: mean x of on pixels in box
7. y-bar: mean y of on pixels in box
8. x2bar: mean x variance
9. y2bar: mean y variance
10. xybar: mean x y correlation
11. x2ybr: mean of x*x*y
12. xy2br: mean of x*y*y
13. x-ege: mean edge count left to right
14. xegvy: correlation of x-ege with y
15. y-ege: mean edge count bottom to top
16. yegxv: correlation of y-ege with x

This allows us to cut out the step of complicated pre-processing and focus on the numerical properties of the images instead. The goal of this project is to be able to correctly classify samples to our target variable:

1. lettr: 26 different capital letters

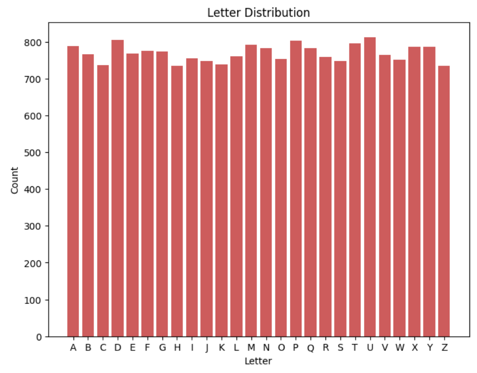


Figure 1: Distribution of letters in our dataset

To prepare the data for modeling, we used the PANDAS library to load the dataset into a DataFrame object. The data was then separated into independent and dependent columns using .drop() from PANDAS.

breakatwhitespace

```python
y = data.letter
x_data = data.drop('letter', axis=1)
```

After, the features (independent variables) were normalized to scale from 0 to 1.

breakatwhitespace

```python
x = (x_data - np.min(x_data, axis = 0)) / (np.max(x_data, axis = 0) - np.min(x_data,
    axis = 0))
```

Finally, train_test_split from SKLEARN.MODEL_SELECTION was used to split our data into a training set and a testing set. For this, we used 80% of our data for the training and 20% for the test.

breakatwhitespace

```python
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state
    =75)
```

3 Modeling

In order to correctly classify the given letters based on their numerical properties, we utilized supervised classification methods. Because of the nature of our data, it seemed fitting to use a supervised approach as our goal was to create a successful model capable of identifying patterns to predict a certain outcome given our pre-labeled training data. Since this project has been undertaken multiple times, we investigate the accuracy of different models and ways to improve them. For this project, we tested three different types of models:

1. Logistic Regression
2. Neural Networks
3. Random Forests

with varying results depending on our model parameters.

3.1 Logistic Regression

For this project, a Multinomial Logistic Regression (using the softmax function) was used. This was done as because we wanted to predict non-binary categorical results. Since we have 26 different classes (26 unique alphabetical letters), our model will estimate coefficients for each class relative to the other 25 classes using the softmax function:

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

where $\vec{z}$ is the input vector and K is the total number (26) of classes.

```python
from sklearn.linear_model import LogisticRegression

#instantiate the LogisticRegression model:
lr = LogisticRegression(multi_class='multinomial', max_iter=2000)

#train it:
lr_model = lr.fit(x_train,y_train)
lr_predictions = lr.predict(x_test)

#calculate the accuracy score when predicting test data:
lr_accuracy = lr.score(x_test,y_test)
```

This model resulted in a mediocre accuracy of 74.93%

3.2 Neural Networks

Multi-layer Perceptrons (MLP) are a type of feedforward neural network that is used on data that is not linearly separable (like ours). It utilizes multiple connected layers that trains using backpropagation algorithm. This is done by calculating gradients of a loss function using the model's parameters then optimizing each parameter by iteratively updating them in order to minimize the loss function.

Our input layer is the same size as the number of features of dataset has. In this case, we have 16 features.

The output layer in our neural network is the same size as the number of classes we are trying to represent. In this case, we have one class for each alphabetical value (26).

The hidden layers in our neural network represent connected neurons that carry weights and biases throughout the network. The number of hidden layers is variable and we explore its optimal value in the next example.

```python
from sklearn.neural_network import MLPClassifier

#instantiate the MLPClassifier model:
mlp = MLPClassifier(hidden_layer_sizes=(200,), max_iter=1000)

```

```python
#train it:
mlp.fit(x_train, y_train)
mlp_predictions = mlp.predict(x_test)

#calculate the accuracy score when predicting the test data:
mlp_accuracy = accuracy_score(y_test, mlp_predictions)
```

This model resulted in an accuracy of 97.03% In our model, we adjusted the number of hidden layers in order to try and further improve our accuracy. It's important to note that while we originally tested with 2000 hidden layers, each incremental increase of the amount of layers changed the accuracy by a marginal amount.

```python
mlp = MLPClassifier(hidden_layer_sizes=(2000,), max_iter=1000)
```

Because of the size of our dataset, we should expect the optimal amount of hidden layers to be quite large. By increasing our training model to support that amount of hidden layers, we would expect the accuracy of our model to increase as well.

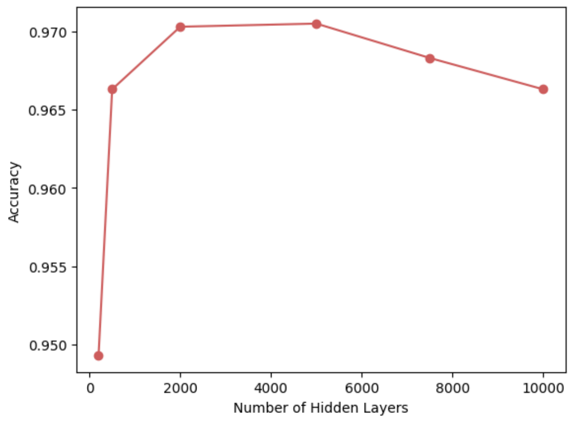


Figure 2: Graph of Hidden Layers vs. Accuracy

Number of hidden layers	Accuracy
200	0.9493
500	0.9663
2000	0.9703
5000	0.9705
7500	0.9683
10000	0.9663

Table 1: Hidden Layers vs. Accuracy

As described above, as we increase the amount of hidden layers our model accuracy increase with it. From our graph, we can actually see that the optimal amount of hidden layers lies between 5000 and 7000 hidden layers. Our accuracy then begins to decrease as we add more hidden layers from that point which may be due to over-fitting.

3.3 Random Forests

Random Forests are an ensemble learning method that utilizes a technique called bagging (bootstrap aggregation). This technique involves training multiple decision trees on samples of random subsets of the data (with replacement) then averages the results of individual decision trees to construct our final model. For this project, the model was built using scikit-learn's RandomForestClassifier:

```python
from sklearn.ensemble import RandomForestClassifier

#instantiate the RandomForest model:
forest_model = RandomForestClassifier(n_estimators=1, random_state=45)

#train it:
forest_model.fit(x_train, y_train)
rf_predictions = forest_model.predict(x_test)

#calculate the accuracy score when predicting test data:
rf_accuracy = accuracy_score(y_test, rf_predictions)
```

In our model, we adjusted the number of trees contained in the forest (n_estimators) to explore how our model accuracy would change as well as search for the optimal amount of estimators.

```
forest_model = RandomForestClassifier(n_estimators = 5000)
```

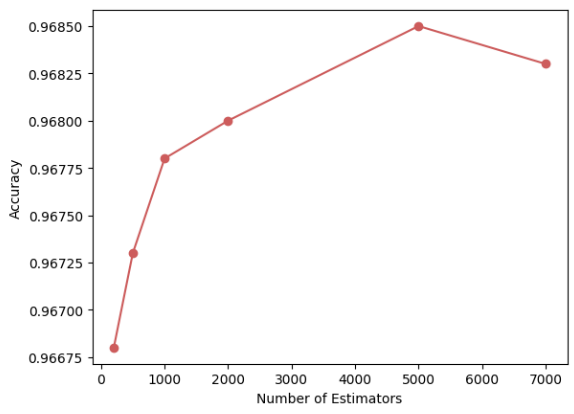


Figure 3: Graph of Estimators vs. Accuracy

Number of estimators	Accuracy
200	0.9668
500	0.9673
1000	0.9678
2000	0.9680
5000	0.9685
7000	0.9683

Table 2: Estimators vs. Accuracy

As shown by the graph, as our number of trees in our Random Forest increases, so does our model accuracy (up to a certain point). When we tested 7000 estimators, our accuracy began to decrease, leading us to believe that the optimal number of trees is most likely between 5000 and 7000 (since that is where the decay begins).

In order to marginally improve our model and attempt to avoid overfitting in our Random Forest, we can employ a method called k-fold cross validation. This approach involves randomly separating our training set into k groups (folds) and fitting a model on each fold. Then we pick the model that performed the best and discard the others. We can iterate this process as many times as necessary.

For this specific project, we will use our Random Forest model with n_estimators $= 5000$ and $k = 5$ and iterate this process 5 times in order to optimize our model.

```
from sklearn.model_selection import KFold
import copy

data_idx = np.arange(len(x_train))
kf = KFold(n_splits=5, shuffle=True)
best_score = -np.inf
best_model = copy.deepcopy(forest_model)

total = 5

for i in range(total):
  k = 0
  print("Attempt {}:".format(i))

  for train_idx, val_idx in kf.split(data_idx):
      x_train_fold = x_train.iloc[train_idx]
      y_train_fold = y_train.iloc[train_idx]
      x_val_fold = x_train.iloc[val_idx]
      y_val_fold = y_train.iloc[val_idx]

      forest_model.fit(x_train_fold, y_train_fold)
      rf_predictions = forest_model.predict(x_val_fold)
      score = accuracy_score(y_val_fold, rf_predictions)

      print("Accuracy in fold {}: {:.5f}".format(k, score))
      print()
      k = k + 1

      if score > best_score:
          best_model = copy.deepcopy(forest_model)
          best_score = score

  print("Best Accuracy:", best_score)
  print()

  forest_model = best_model
```

Our results from the cross validation are as follows:

Fold	Attempt 1	Attempt 2	Attempt 3	Attempt 4	Attempt 5
1	0.9484	0.9575	0.9609	0.9653	0.9616
2	0.9638	0.9591	0.9581	0.9581	0.9578
3	0.9650	0.9603	0.9644	0.9594	0.9597
4	0.9550	0.9634	0.9628	0.9563	0.9544
5	0.9634	0.9628	0.9653	0.9622	0.9606

When using our cross-validated model on the test data, we receive an accuracy of 95.98%. It seems as our model's accuracy plateaus around 96% and there is only so much we can do to improve it at the margin without overfitting. Compared to our original model's accuracy of 96.85%, our cross-validated model's accuracy is lower which could indicate small signs of overfitting in our original Random Forest model.

4 Results

When comparing the accuracies of our 3 models, we can see that the Random Forest and Neural Network both performed extremely well compared to the Logistic Regression.

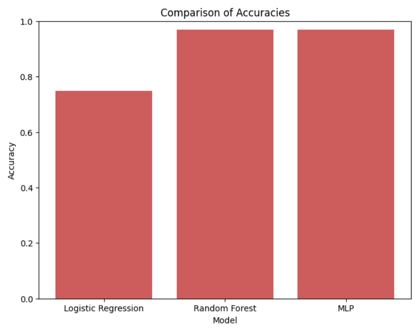
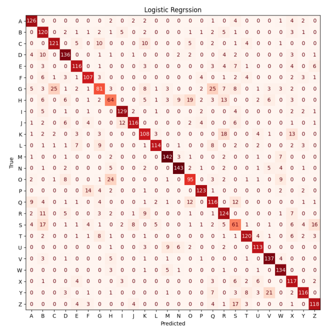


Figure 4: Bar graph of model accuracies

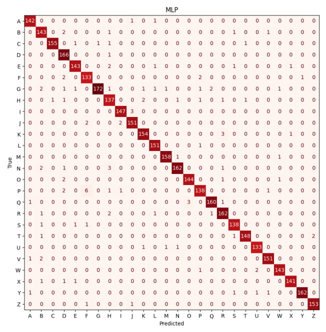
Model	Accuracy
Logistic Regression	0.7493
MLP	0.9705
Random Forest	0.9685

Table 3: Table of model accuracies

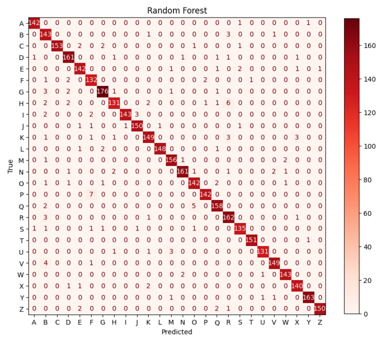
Plotted below are the three confusion matrices for each of our models.



(a)



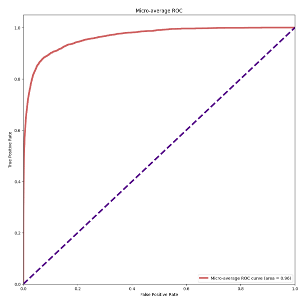
(b)



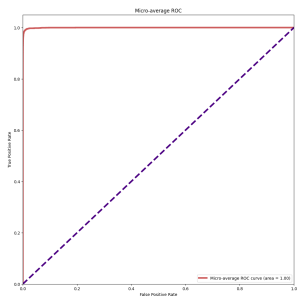
(c)

Figure 5: (a) Logistic Regression (b) MLP (c) Random Forest

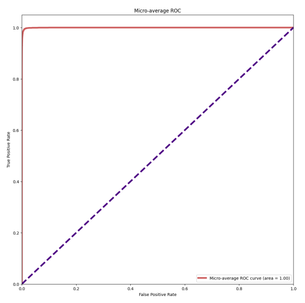
Next, we can examine the ROC curves. Since our project involves non-binary multiple classes, we plotted the micro-average ROC curve for each model using the OneVsRestClassifier from SKLEARN.MULTICLASS. We started by first computing the ROC curve for each individual letter. Then, we can aggregate the scores from each individual class to compute an average based on the weight each class has on the data (in our case, the distribution of letters is almost uniform, but we can still use this technique to account for any discrepancies).



(a)



(b)



(c)

Figure 6: (a) Logistic Regression (b) MLP (c) Random Forest

5 Discussion

From our confusion matrices and ROC curves, we can see the both the Random Forest and MLP models performed incredibly well. On the other hand, our Logistic Regression model performed mediocre at best, misclassifying letters such as O and G.

After determining the accuracies of each model, we wanted to see how each model was actually visualizing the numerical values of each letter. To do this, we plotted an actual letter from our dataset against each of the model's visual interpretation of that same letter using MATPLOTLIB's .imshow() function.

```python
import numpy as np
import matplotlib.pyplot as plt

models = {
    'Logistic Regression': lr_y_pred,
    'Random Forest': rf_y_pred,
    'MLP': mlp_y_pred
}

letter = 'A'
fig, axes = plt.subplots(1, 4, figsize=(15, 4))
fig.suptitle(f"Predictions for Letter '{letter}'", fontsize=16)

actual_letter_index = np.where(y_test == letter)[0][0]
actual_image = x_test.iloc[actual_letter_index, :].values.reshape(4, 4)

axes[0].imshow(actual_image, cmap='binary', interpolation='nearest')
axes[0].set_title("Actual")
axes[0].axis('off')

for i, (model_name, y_pred) in enumerate(models.items()):
    letter_indices = np.where(y_pred == letter)[0]
    if len(letter_indices) > 0:
        example_image = x_test.iloc[letter_indices[0], :].values.reshape(4, 4)
        axes[i + 1].imshow(example_image, cmap='binary', interpolation='nearest')
        axes[i + 1].set_title(model_name)
        axes[i + 1].axis('off')

plt.tight_layout()
plt.show()
```

Since we have 16 features, we are taking a one-dimensional array of those features and reshaping it into a two-dimensional 4x4 array for our chosen letter.

For example, the letter A:

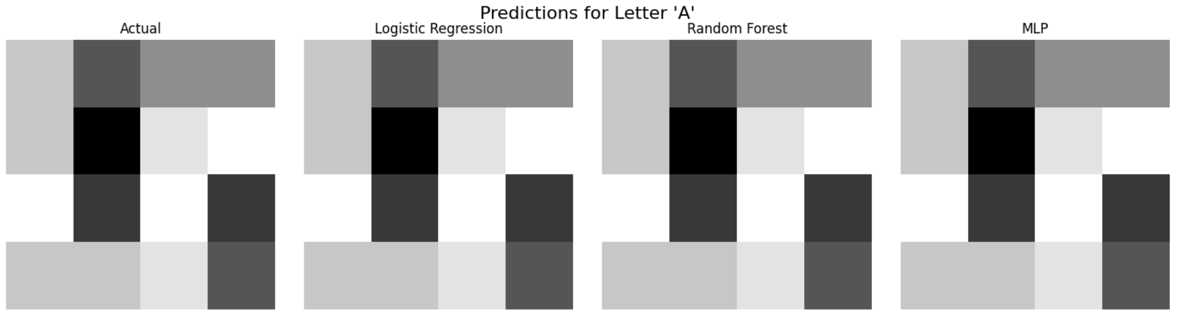


Figure 7: Visualization of the letter A

While this obviously does not look anything remotely close to the letter A, this is because we are still using the numerical values of the dataset and not recreating the actual image itself. Note that our numerical values come from features such as:

1. x-box: horizontal position of box

2. y-box: vertical position of box

Regardless, all our models performed well on the letter A, matching the actual interpretation perfectly. Now, let's look at an example where our models did not perform as well.

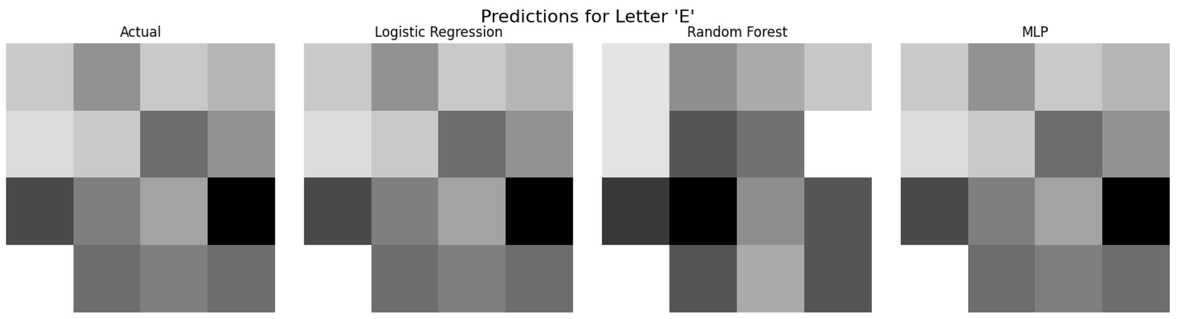


Figure 8: Visualization of the letter E

In this example, our near-perfect Random Forest model incorrectly visualizes the letter E. Surprisingly, our less accurate Logistic Regression model correctly visualizes it.

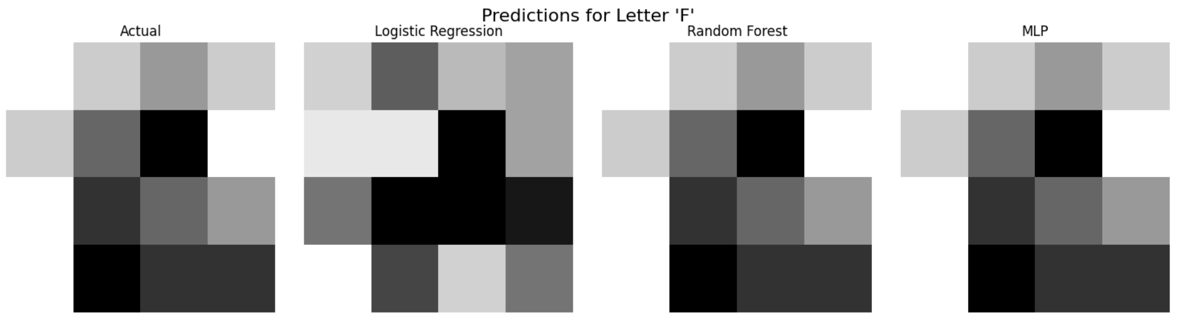


Figure 9: Visualization of the letter F

In this example, our Logistic Regression model very much incorrectly visualizes the letter F.

We can guess that the reason for the Random Forest's incorrect visualization is due to the complexity of its decision boundaries. Since our Random Forest model used so many trees, we saw it was more prone to over-fitting (as shown in our cross-validation example above). Since the logistic regression is a bit simpler, it could be less prone to over-fitting in this example.

6 Conclusion

In summary, the purpose of this project was to investigate different types of machine learning techniques applied to this OCR problem.

We conclude that many different models can be used to analyze this data and complete this classification task. From our results, we can see that each model has its own strengths and weaknesses due to over-complexity or simplifications.

For example, our Logistic Regression model performs badly compared to the other models we tested. This may be due to the fact that the relationship between our input variables and target variables were not linearly separable.

However, the model we found to work the best was our Random Forest model. While their accuracies were similar, the Random Forest being lower than MLP, the time it took to run each model varied drastically. For our final models, it took the Random Forest around 3 minutes to complete training and testing over our entire dataset. Comparatively, it took our MLP model 40 minutes to do the same thing. While our MLP model may have been marginally more accurate in the end, the trade-off between efficiency and accuracy seemed appropriate enough to pick the Random Forest model over the MLP. Along with this, our MLP model had never been tested with our k-fold cross validation so there may be over-fitting factors that have not been taken into account.

For future projects, we can develop our work by carefully accounting for over-fitting and the non-linear relationships between each feature. Obviously, this makes sense as alphabetical letters are not unique in the numerical values we were provided in this dataset and are definitely correlated with one another. To truly avoid this problem, we could employ computer-vision techniques in order to define the numerical pre-processing ourselves and train our model on the actual image instead of its strictly numerical attributes. This would most likely improve the accuracy of each of our models yet loses the efficiency we gain by just having to analyze numbers.

7 References

1. S. Karthikeyan, A. G. S. de Herrera, F. Doctor and A. Mirza, "An OCR Post-Correction Approach Using Deep Learning for Processing Medical Reports," in IEEE Transactions on Circuits and Systems for Video Technology, vol. 32, no. 5, pp. 2574-2581, May 2022, doi: 10.1109/TCSVT.2021.3087641

2. P. L. Lekshmy, S. Velmurugan, I. Kumari, S. Kayalvili, B. T. Sree and P. K. Kumar, "Optical Character Recognition (OCR) in Handwritten Characters Using Convolutional Neural Networks to Assist in Exam Reader System," 2024 2nd International Conference on Advancement in Computation and Computer Technologies (InCACCT), Gharuan, India, 2024, pp. 623-627, doi: 10.1109/InCACCT61598.2024.10551027

3. Slate, D. (1991). Letter Recognition [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C5ZP40.